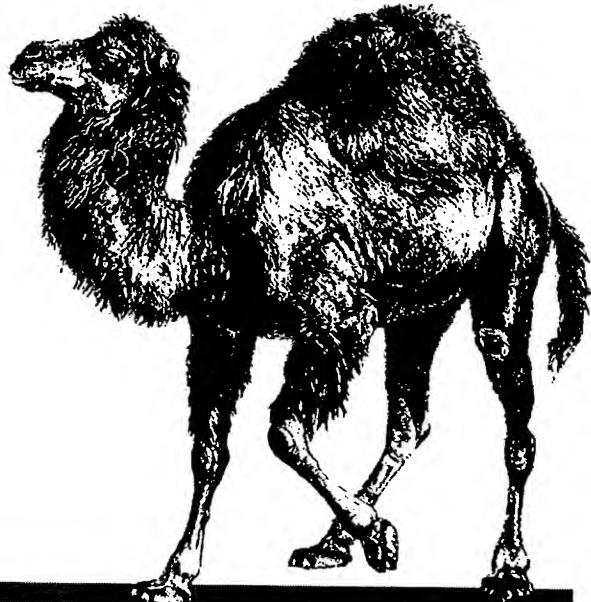


*UNIX Programming*  
**Perlプログラミング**



*Programming*

---

**perl**

*Larry Wall and Randal L. Schwartz* 著  
近藤 嘉雪 訳



ソフトバンク株式会社

シェルでは、シングルクオートの内側以外のあらゆる場所で変数展開が行われるが、Perlでは、定められた場所——ダブルクオートの内側など——でしか変数展開（および\nなどの逆スラッシュの解釈）は行かない。バッククオートで囲んだ文字列でも、ダブルクオートのときと同様に変数展開（逆スラッシュの解釈）が行われるが、バッククオートの場合は、変数展開で得られた文字列をさらにコマンドとして実行して、その結果得られた出力が最終的な値となる：

```
$lines = 'wc -1 $filename';
```

（注意：ルリや出力をリダイレクトする必要がなく、ただコマンドを実行したいだけなら、system関数を使うこと。第4章のsystem()関数（260ページ）を見よ。）また、次節で紹介するバターンマッチでも、変数展開が行わる。変数名の直後の文字が、変数名の続きに見えるね（それがあるときには、変数名をアレースで囲む必要がある：

```
/the merry ${month} of May; # 変数$moが展開され,  
# nthはそのまま残される。
```

この例で、ブレースがないと変数\$monthの値が展開されてしまう。

## 1.4 ■ パターンマッチ

Perlには、m//とs///の2種類のパターンマッチ演算子（pattern matching operator）がある。これらは、それぞれマッチ演算子（match operator）、置換演算子（substitution operator）と呼ばれる（スラッシュの代わりに、英数字以外の任意の文字を使うことができる）。スラッシュを~~（リミテッド）~~に使うときには、m//演算子は、單に//演算子と呼ばれる（そのためには、m//演算子は、單に//演算子と呼ばれることがある（UNIXプログラムの多くでは、テリミタとしてスラッシュしか使えないのも、その理由の1つである）。m//演算子は、テリミタ間に指定されたパターンを文字列の中央から探し、マッチした（ハターンを含んでいる）かどうかによって、真または偽を返す。s///演算子も同じ処理を行うが、文字列のうちマッチした部分を、2番目と3番目のテリミタの間に指定されているもので置き換える。パターンは、正規表現（regular expression）を使って指定する。おそらく読者の多くは、sedやviなどのUNIXツールをはじて、すでに正規表現に親しんでいること

とだろう。しかしPerlが扱う正規表現は、他の正規表現ほとんどのスケーリングになっているので、いざれにせよ次の一覧表を注意深く見てほしい：

改行以外の任意の1文字にマッチ	\n
集合に含まれるいずれか1文字にマッチ	[a-z0-9]
集合に含まれない文字1つにマッチ	[^a-z0-9]
数字1文字にマッチ。[0-9]と同じ	\d
英数字（alphanumeric：単語を構成する文字）の1文字にマッチ	\D
英数字以外の1文字にマッチ。[~0-9]と同じ	\W
空白文字（スペース、タブ、改行など）の1文字にマッチ	\s
空白文字以外の1文字にマッチ	\S
改行文字（newline）にマッチ	\n
復帰文字（return）にマッチ	\r
タブ（tab）にマッチ	\t
改ページ文字（form feed）にマッチ	\f
バックスペース（backspace）にマッチ	\b
ヌル文字（null character）にマッチ	\0
これもヌル文字にマッチする。なぜなら、……	\000
逃逸（escape）文字のASCII文字にマッチ	\nnn
16進値nnのASCII文字にマッチ	\xnn
ASCIIコントロール文字にマッチ	\cN
その文字自身にマッチ（\l, \L, \*など）	\mctachar
後方参照するためにマッチを記録しておく	\\$ (abc)
最初のカッコがマッチしたものにマッチ	\1
2番目のカッコがマッチしたものにマッチ	\2
以下同様……	\3
0または1個のxにマッチ。xは[は]記のいぢれか	\?

```

x* 0 個以上の x にマッチ
x+ 1 個以上の x にマッチ
x{m,n} m 個以上 n 個以下の x にマッチ

abc a, b, c にこの順番でマッチ
fee|fee fee, fie, foe のどれか 1 つにマッチ

```

```

\b 単語の境界にマッチ ([] の外側のみ)
\B 単語の境界以外にマッチ
^ 行または文字列の先頭にマッチ
$ 行または文字列の末尾にマッチ

```

他の正規表現とは違って、Perl の正規表現では、メタキャラクタとして使うときは、カッコ、アラカルト、縦棒の前に逆スラッシュを置かなければならない。反対に、普通の文字として使うときは、これらの直前に逆スラッシュを置かなければならない。(このような逆スラッシュの扱いには、それなりの理由がある。これらの文字はほとんどの場合メタキャラクタとして使われるのでもしメタキャラクタとして使うときに逆スラッシュ) また、この後すぐお話しするように、メタキャラクタを簡単には書きたいという相点もある) このほかにも、数字、英数字、空白文字を表す略記法が用意されている。

他の正規表現と同じように、パターンがカッコを含んでいるとき、それより後ろの部分では、カッコの前にマッチしたものを \1, \2, \3……[1]によって参照することができます(数字は、パターンの左端から数えて、何個目の左カッコかを示す)。M, \2……のことを後方参照 (backreference) と呼ぶ。後方参照は、カッコで囲まれたパターンではなく、カッコが実際にマッチしたものを持つことに注意しよう。後方参照は、まったく同じ文字の並びにマッチしなければならない。10ページで取り上げた baregrep プログラムを、次のように起動したとしよう：

```

baregrep '(\w+)\s*=\s*\1' *.c

```

この結果、C ソースファイルのうち、hump = hump のように、変数に自身を代入している行がすべて表示される。(ところで、この例では `hump = hump` のような

行も表示されてしまう。なぜだろうか? ヒント: \b は単語の境界にマッチする。) マッチした文字列のうち、カッコで囲まれた部分を部分文字列<sup>15)</sup> (substring) といふ。多くの言語では、パターンや置き換え文字列の中で部分文字列を後方参照することができるが、Perl はさらに一步先をいっている。パターンマッチが成功すると、すべての部分文字列は特殊変数 \$1, \$2, \$3……に残されるようになっている。これらの変数には、後方参照と同じ番号がつけられている。後方参照よりも、\$1 のような変数を使うことが推奨される。(パターン自身の中は別である。なぜなら、パターンマッチの途中には、これらの変数はまだ作られていないから) したがって、置換演算子を例にとると、置き換え文字列の中では \1, \2 の代わりに、\$1, \$2 を使うほうがよい。

ところで前節では、ダブルクオートで囲んだ文字列と同じように、パターンも変数展開の対象になると説明した。パターンの中では、\$ は、行末 (または文列の末尾) を表さない場合に限って、変数展開の対象になる：

```

/the merry month of $mo/; # 変数 $mo の値を展開する。
/the merry month of May$/; # $ は行末にマッチする。

```

同様に、置換演算子では、パターン、置き換え文字列とともに変数展開の対象になる：

```

s/$monthname/$monthnum/; # April を 4 へ置換、といったことをする。
s/\r\n$/\n/; # 行末の CRLF を LF に置換する。
s/(\S+)\s*+\1/$1/; # 最初の 2 つの単語を入れ換える。

```

変数展開はパターンをコンパイルする前に行われる。正規表現ハーサには、変数展開後のパターン文字列が置きられるので、パターンに変数が含まれていたことは正規表現ハーサではない。それゆえ、変数の値に含まれるすべてのメタキャラクタは、(普通の文字ではなく) メタキャラクタとして認識されてしまう。変数に含まれるメタキャラクタを解釈しない(普通の文字として扱う) ためには、次のようにして英数字以外の文字すべての前に、逆スラッシュを挿入すればよい：

```

$string =~ s/(\W)/\\$1/g;
/a pattern containing a $string to be interpreted literally/;

```

<sup>15)</sup> [訳注] substr 関数によって文字列が取り出したものも部分文字列と呼ぶので、混同しないように気をつけよう。たいていは文脈から区別できる。